Reinforcement Learning for Water Management

1 Introduction

Scarcity of water and the increasing awareness of the need to save energy in providing good quality water to increasing numbers are driving the search for new ways to save water as well as energy and improve the financials of water utilities. At the same time the increasing "digitalization" of urban Water Distribution Networks (WDNs) is generating huge amounts of data from flow/pressure sensors and smart metering of household consumption and enabling new ways to achieve more efficient operations. Sequential decision models are offering an optimization framework more suitable to capture the value hidden in real time data assets. More recently, a sequential optimization method based on Approximate Dynamic Programming (ADP) has been proposed, whose preliminary computational results demonstrate that this methodology can reduce the electricity expenses while keeping the water pressure in a controlled range and, at the same time, is able to effectively deal with the uncertainty on the water demand.

1.1 Pump Scheduling Optimization

Optimization of WDNs has been a very important field in the Operation Research community at least in the last 40 years and many tools from Mathematical Programming as well as metaheuristics have been proposed. An updated review on optimization of water distribution systems is given in [1], a very recent, wide and systematic survey where several classes of existing solutions including: linear programming, nonlinear programming, dynamic programming, metamodeling, heuristics, and metaheuristics are deeply analysed and referenced.

One of the major issues for water utilities is the excessive energy consumption due to non-optimal pumping scheduling, usually known as Pump Scheduling Optimization (PSO) problem. A pump schedule defines which pumps are to be operated and with which settings at different periods of the day. PSO must be performed by also considering operational constraints, such as: satisfying demand, keeping pressures within certain bounds to reduce leakage and the risk of pipe burst, and keeping reservoir levels within bounds to avoid overflow.

One of the earliest approaches for PSO was based on Dynamic Programming (DP) [2]. However, since the number of states increases exponentially with the size of the WDN, this type of solutions has been usually considered impractical, due to the curse of dimensionality in DP: recently, [3] approximate methods has been proposed to overcome the curse of dimensionality.

A formulation closely related to DP is that based on a Markov Decision Process (MDP). In [4, 5], an MDP was used to model the PSO problem in the case of a simplified WDN with three water reservoirs. The disadvantage of using classical MDP is the need to control the size of the state space through coarser discretization, while a major advantage is that the solution provides full spectra of possible policies from whichever initial stage (i.e. defined through various discrete levels of reservoirs in the WDN). This is essentially a planning approach [6] that requires extensive simulations for building the state transition probability matrix. For instance, for each state-action pair in [5] more than a hundred random runs needed to be performed. A similar strategy was recently proposed in [7, 8], where the MDP is generated through an exhaustive interaction with the hydraulic simulation software and DP is used to solve the PSO problem.

An Approximate Dynamic Programming (ADP) framework to PSO, based on MDP models but without the requirement of complete knowledge of transitional dynamics, has been recently proposed in [9]. The basic idea is that, instead of generating the complete MDP of the WDN behaviour – and then solve the PSO using DP – exploration (i.e. try pumps configurations not yet evaluated with respect to the current state of the WDN) and exploitation (i.e. exploit available knowledge acquired so far, relatively to the current state of the WDN) are alternated. Among ADP strategies, Q-Learning is specifically considered, a well-known algorithm in the Reinforcement Learning (RL) community. The proposed approach has been tested on a well-known benchmark WDN, namely Anytown, first proposed for PSO in [10].

2 Markov Decision Processes

MDPs are a powerful framework that can be applied to model a variety of sequential optimization problems in different fields [11]. It works in a sequential process of decision epochs by performing actions that change the state at the next decision epoch, accordingly to a transition probability function representing the dynamics of the system, and that provides for the performed action [12]. Since the system is ongoing, the state of the system prior to next decision depends on the present decision/action. Therefore, the goal is to identify, for each state, the action that produces the highest expected reward in a long-time horizon and that will result in the system performing optimally with respect to some predetermined performance criterion.

In a formal way an MDP can be defined as a tuple $< S, \mathcal{A}, \mathcal{T}, \mathcal{R} >$ where S is a set of discrete states, $\mathcal{A}$ is a set of discrete actions, $\mathcal{T}$ is a state transition function, and $\mathcal{R}$ is a reward function [6]. The set of discrete states is defined as the finite set $S = \{s_1, s_2, \ldots., s_{N_S}\}$ where the size of the state space is N_S. Each state consists in N_V state variables, such that $s = \{\xi^1, \xi^1, \ldots.,, \xi^{N_V}\}$. Thus, the set of discrete states has cardinality N_S and each state is composed by the tuple of N_V state variables, as follows:

$$S = \{s_1, \ldots., s_{N_S}\} = \{(\xi_1^1, \ldots., \xi_1^{N_V}), \ldots, (\xi_{N_S}^1, \ldots., \xi_{N_S}^{N_V})\} \quad (1)$$

On the other hand, the actions allow for moving from one state to another. The actions set, $\mathcal{A}$, can be defined as the union of all the subsets of allowed actions for every state s of the state set S:

$$\mathcal{A} = \cup_{s \in S} \mathcal{A}_s \quad (2)$$

By applying an action $a_t = a$ in the state $s_t = s$ at time step *t* the system makes a transition to the new state $s_{t+1} = s'$ based on a probability distribution $\mathcal{T}$ over the set of possible transitions:

$$\mathcal{T}(s,a,s') = P(s_{t+1} = s' | s_t = s, a_t = a) \qquad (3)$$

where:

$$\mathcal{T}: S \times \mathcal{A} \times S \to [0,1], \textstyle\sum_{s' \in S} \mathcal{T}(s,a,s') = 1 \qquad (4)$$

and

$$0 \le \mathcal{T}(s,a,s') \le 1 \qquad (5)$$

Finally, the reward function specifies a value received for performing the action $a_t = a$ in the state $s_t = s$ at time step *t* and is defined as:

$$\mathcal{R}(s,a,s') = \{r_t | s_t = s, a_t = a, s_{t+1} = s'\} \qquad (6)$$

Where $\mathcal{R}: S \times \mathcal{A} \times S \to \mathbb{R}$: while a positive $\mathcal{R}(s,a,s')$ may be regarded as a reward, a negative one can be considered a cost/punishment.

Solving an MDP consists in finding a control policy π, which is defined as a mapping from states to actions, $\pi: S \to \mathcal{A}$. Optimizing such policy corresponds to maximize the accumulated reward values received over a long-time horizon. To achieve this goal, the definition of value function or utility must be provided. The value function of a state $s_t = s$ at time step *t* under control policy π, denoted by $V^{\pi}(\sigma)$, is the expected return of rewards when starting in state *s* and by following, sequentially, the actions suggested by the policy π. The value function is therefore defined as:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[\textstyle\sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k} | s_t = s] \qquad (7)$$

where $\mathbb{E}_{\pi}$ is the expected value given by following policy π, and $\gamma \in [0,1]$ is a discount factor that is used to balance current and future rewards. When γ is small the approach is said to be "myopic", which means that it is only concerned about immediate rewards, while, when γ is large future rewards become also important.

2.1 Dynamic Programming

One fundamental property of a value function is that it satisfies the Bellman Equation [13], that allows to break a dynamic optimization problem into simpler sub-problems, and can be defined recursively as follows:

$$\begin{aligned} V^{\pi} &= \mathbb{E}_{\pi}\{r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \ldots | s_t = s\} \\ &= \mathbb{E}_{\pi}\{r_t + \gamma \cdot V^{\pi}(s') | s_t = s, s_{t+1} = s'\} \\ &= \mathcal{R}(s,\pi(s),s') + \gamma \sum_{s' \in S} \mathcal{T}(s,\pi(s),s') \cdot V^{\pi}(s') \end{aligned} \qquad (8)$$

An optimal policy, denoted by π^*, is such that $V^{\pi^*}(s) \ge V^{\pi}(s)$ for all $s \in S$ and all policies π. Thus, the optimal value function can be evaluated as follows:

$$V^*(s) = \max_{a \in \mathcal{A}} [\mathcal{R}(s,a,s') + \gamma \sum_{s' \in S} \mathcal{T}(s,a,s') \cdot V^*(s')] \tag{9}$$

Finally, the optimal policy π^* consists in the optimal actions selected according to the optimal value function V^* and it is summarized by:

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} [\mathcal{R}(s,a,s') + \gamma \sum_{s' \in S} \mathcal{T}(s,a,s') \cdot V^*(s')] \tag{10}$$

There are two algorithms to compute the optimal policy π^* : *policy iteration* [12] and *value iteration* [6]. The former computes the utility of all states and improves the policy in each iteration until actions convergence to an optimal policy; the latter computes the expected utility of each state using the utilities of the neighbour states until the utilities for two consecutive steps are close enough.

Finally, independently on the specific algorithm, the best action will be the one with the highest expected value based on possible next states resulting from taking that action [14].

DP is also an effective approach for stochastic problems. The main difference is the need to model that information becomes available after the action a is performed, so uncertainty is both in next reached state s' and in the reward.

The deterministic form of the Bellman's equation can be adapted to the stochastic case by simply replacing the (deterministic) transition matrix with a probability transition matrix:

$$V^*(s) = \max_{a \in \mathcal{A}} [\mathcal{R}(s,a,s') + \gamma \sum_{s' \in S} \mathbb{P}(s'|s,a) \cdot V^*(s')] \tag{11}$$

this is also known as the standard form of the Bellman's equation, used in almost every textbook on stochastic programming and dynamic programming.

Finally, if expected value is used instead of the sum of probabilities, an equivalent form – namely the expectation form of the Bellman's equation – is obtained, which is more appropriate for ADP:

$$V^*(s) = \max_{a \in \mathcal{A}} [\mathcal{R}(s,a,s') + \gamma \mathbb{E}\{V(s')|s\}] \tag{12}$$

2.2 Approximate Dynamic Programming

ADP offers a powerful set of strategies for solving problems that are large as well as small but lacking a formal model, specifically the transition function. Most of relevant real-life problems belong to this class, also known as "information acquisition", where performing an action is the only way to obtain a better estimate of its value and increasing the knowledge about the system. While exact DP steps backward in time, computing the value function then used to produce optimal decisions, ADP steps forward in time, so an approximation of the value function updated and used to make decisions. Basically, going forward in time requires to alternate between:

- randomly generating a sample of what might happen

- making a decision about the action to perform.

According to [15], sampling can be performed in three different ways: from real data (e.g. real physical processes), via computer-simulation or sampling from a known distribution. The main benefit of ADP algorithms is the ability to solve problems without knowing the underlying probability distribution, which is usually the case of working on real data. On the other hand, when a probability model is available Monte Carlo simulation can be used to generate samples; another possibility is to use software simulation of the physical system to perform sampling. However, physical systems are usually complex and difficult to be mathematically modelled and probability distributions and/or simulation are not available: the system's behavior, in terms of transitions and reward, can only be observed by direct interaction, while optimization is performed accordingly to the expectation form of the Bellman's equation (12).

3 PSO as a learning problem

The value function approximations known as "lookup tables" allow to store a value $V(s)$ for each discrete state $s \in S$, in the case of state value function. Another possibility is to use the state-action value function, which represents how good is to perform action a in state s, for every pair state-action. More specifically, an ADP strategy of the latter type is used, known as Q-learning, a popular approach used for problems with small state and action spaces, where we do not have a mathematical model for how the system evolves over time.

3.1 Q-Learning

Q-Learning takes its name from the variable $Q(s, a)$ which is the value of being in the state $s \in S$ and taking the action $a \in \mathcal{A}$ (i.e. state-action value function).

Another typical classification adopted in the RL community is between "learning" and "planning" algorithms. While planning algorithms have access to a model/simulator of the world, learning algorithms do not know anything about the system dynamics and must learn how to behave by direct experience with it (aka "environment"). Furthermore, RL algorithms can be used to address optimization problems by exploiting a "learning-by-doing" paradigm in two different ways:

- apply a planning algorithm after that a model of the world has been learned from experience (model-based approaches)
- learn and apply a policy or value function directly from experience (model-free).

Q-learning belongs to the latter category and, as an ADP algorithm, it works going forward in time, where the next action to perform is selected according to:

$$a^n = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \bar{Q}^{n-1}(s^n, a) \tag{13}$$

where $\bar{Q}^n(s^n, a)$ is an estimation of the true value of $Q(s, a)$ after n iterations, a^n and s^n are, respectively, the action to choose and the state the system is at the n-th iteration. Thus, at iteration n the estimation of $Q(s, a)$ at iteration $n-1$, namely $\bar{Q}^{n-1}(s', a')$, is used.

After the action a^n, an immediate reward r is observed along with the new state s', so the value of the state-action pair, $\bar{Q}^n(s^n, a^n)$, is updated consequently:

$$\bar{Q}^n(s^n, a^n) = \bar{Q}^{n-1}(s^n, a^n) + \alpha \left[r + \gamma \max_{a' \in \mathcal{A}} \bar{Q}^{n-1}(s', a') - \bar{Q}^{n-1}(s^n, a^n)\right] \qquad (14)$$

Where α is the learning rate – which set how much the old estimate of the Q-value has to change depending on the observed state and reward – and γ is the discount factor as defined in the Bellman's equation. Learning rate should be slightly decreased over time to guarantee convergence to the true Q-value.

In this updating rule, the immediate reward plus the discounted max Q-value in the observed next state is what satisfies the Bellman equation (expectation form eq. 12).

Selecting the action a^n does not guarantee to reach an optimal solution: this is just one component of the Q-Learning algorithm which is basically associated to exploitation (i.e. making decisions depending on the knowledge acquired so far). As already mentioned, the goal of Q-Learning is to alternate optimization – thus exploitation – with learning something new about the environment – thus exploration. This is exactly the behavior required to an ADP approach to step forward in time balancing between: *(i)* randomly generating a sample of what might happen (exploration) and *(ii)* making a decision about the action to perform (exploitation).

In Q-Learning a typical strategy to manage this trade-off is known as ε-greedy policy, where the action a^n is randomly selected among all the possible actions, with probability ε, and "greedily" (according to eq. 13), with probability 1- ε.

Solving the problem of when to explore and when to exploit is known as the exploration versus exploitation dilemma. This is a difficult problem and an active area of research. Not surprisingly, Q-learning is difficult to apply to problems with even modest state and action spaces, but its value lies in its ability to solve problems without a model and to work online with the system (learning-and-optimization).

3.2 An MDP for PSO: EPANET simulator as an Environment

To solve PSO through RL, the first step consists in defining the underlying MDP. A similar approach has been proposed in [16] where the goal is to minimize energy cost over a certain time-period (horizon) under constraints on operation parameters. In [9], authors decided to model the set of possible state, $s \in S$, through 2 variables: level of tank (h) and average pressure (p):

$$s = (h_t, p_t) \qquad (15)$$

Differently from other approaches, such as in [7], they do not include time T as state variable. Another crucial difference is that

- in [7] the MDP is generated exhaustively by interacting with EPANET, so including time in the definition of state helps in modelling the deterministic transitions of the WDN system. Then DP – specifically, value iteration – is used to solve the MDP, assuming the learned policy is optimal for the WDN.
- instead, in [9], ADP is used and the MDP is not generated exhaustively (specifically the transition dynamics) but exploration and exploitation are performed to learn something more about the MDP by interacting with the system (exploration) while trying to generate the optimal strategy (exploitation).

Thus, while in [7] EPANET is used a simulator for generating the MDP, in [9] EPANET is assumed to be a real WDN, whose real data are acquired through sensors. Since Q-Learning is based on the expectation form of the Bellman's equation, the resulting policy should be more robust to uncertainty – basically related to water demand – with respect to the policy obtained by solving the deterministic MDP through DP.

Both the two state variables are continuous, and discretization is required to work with tabular MDP. For instance, a discretization on 5 levels, for both h and p, results in $5 \times 5 = 25$ possible states.

The set of actions $\mathcal{A}$ is represented by the variables that can affect the system's state. With respect to WDN, actions are represented by the status of the pumps, so that a generic action $a \in \mathcal{A}$ is defined as follows:

$$a = \{u_1, u_2, \dots, u_{N_U}\} \tag{16}$$

where u_j is the status of the j-th pump in the system and $1 \leq j \leq N_U$ and N_U is the overall number of pumps (i.e. $N_U = 4$ in the case study considered in [9]). When only on/off pumps are considered, the possible values for every component of the action is $u_j = \{0,1\}$, where 0 represents pump *off*, 1 represents pump *on*.

The transition function $\mathcal{T}$ in a WDN cannot be formally defined due to recurrence relation between the variables that compose the system. Thus, the transition function can be described as:

$$(h_{t+1}, p_{t+1}) = f(h_t, p_t, a_t) \tag{17}$$

In actual system h and p are provided by sensors, while in our experiments they are computed by EPANET. An important exogenous variable is the water demand, where d_t is the demand from t to $t+1$: it is unknown to the optimization algorithm. Water demand is a fundamental input for EPANET as it drives – exactly as a in real world WDN – the hydraulic behavior of the network. Since d_t can be observed only at $t + 1$, it cannot be included in the definition of s_t; thus, reward associated to a_t is computed at $t + 1$ and depends on s_t as well as the exogenous variable d_t and the action a_t.

Reward function is defined according to the PSO goal, which is to identify the pump schedule that minimizes the energy cost while satisfying the water demand. The objective function of the PSO problem, in the ON/OFF pumps setting, can be formulated as follows. Let denote with C_t the energy costs from $t-1$ to t, resulting form the configuration of pumps at $t-1$. Thus, C_t depends on the decision variables $a_{t-1} \in \{0,1\}^{N_U}$, with N_U the number of pumps (maintaining the same notation used for actions). The final goal is to identify the actions a_t, with $t = 1,..,T$, that minimizes the total energy cost:

$$\min \sum_{t=1}^{T} C_t \tag{18}$$

To define C_t, is important to highlight that two consecutive time steps in the ADP algorithm can be related to many EPANET simulation time steps. For instance, in our case study simulation steps are on hourly basis while time steps for ADP are specific hours of the day when the action – i.e. possible modification in the activation of the pumps – is performed. So, we need to introduce an operator, $\varphi(t)$, which maps a time step of the ADP to a time step of the simulation. Just a simple example: suppose to have just two time steps in the ADP algorithm, which refer to the 6th and 18th time steps of the simulation, respectively, then $\varphi(t=1) = 6$ while $\varphi(t=2) = 18$. This notation allows for a definition of C_t which considers the simulation time resolution and, therefore, hourly variations in the price of energy.

$$C_t = \sum_{i=\varphi(t-1)}^{\varphi(t)} c_i \left(\sum_{j=1}^{N_U} Q_{i,j} \frac{H_{i,j}}{\eta_j} a_{t,j} \right) \tag{19}$$

Where:

- c_i is the energy price at time i (i.e., i-th hour of the day in a typical PSO setting)
- $Q_{i,j}$ is the quantity of water provided by the j-th pump at time i
- $H_{i,j}$ is the head loss of the j-th pump at time i
- η_j is the efficiency of the j-th pump (it does not depend on time)
- $a_{t,j}$ is the status of the j-th pump from $t-1$ to t (which is constant for every $i \in [\varphi(t-1);\ \varphi(t)]$)

In any case, the computation of C_t is performed by EPANET and the result is provided at the end of the simulation. Finally, the reward function $\mathcal{R}$ can be described as:

$$\mathcal{R} = \begin{cases} r_t = 0 & if\ t = 0 \\ r_t = |\bar{C}_t - \bar{C}_{t-1}| & if(t > 0 \wedge C_{t-1} < 10^9 \wedge C_t < 10^9) \\ r_t = 10^9 & otherwise \end{cases} \tag{20}$$

Where *t* is the time step over the optimization process, starting from *t*=1 to *t*=*T*, C_t represents the energy cost at time *t* while $\bar{C}_t$ is the cost associated to the pump schedule up to *t*, as computed by EPANET.

In [9], authors reported that the most important result is that all the schedules generated by the ADP approach are feasible when the strategy learned by the RL agent is applied to new scenarios characterized by variation of the water demand. Naturally, the policy on these new scenarios is not able to provide the optimal (deterministic) solution, but in any case, a feasible – and quite good – schedule even under uncertainty of the water demand.

4 Concluding remarks

The increasing "digitalization" of urban water distribution networks is generating huge amounts of real time data from flow/pressure sensors and smart metering of household consumption and enabling new ways to achieve more efficient operations. Approximate Dynamic Programming has the potential to leverage the value hidden in real time data assets into energy driven PSO which satisfies the operational constraints. The main result is that ADP strategies are robust with respect to different demand level. Thus, while traditional approaches require to know the water demand in advance or, at least, to have a reliable and accurate forecasting, ADP provides a policy, that is a strategy to decide how to act, as new sensor data become available. Future work will have to explore other Reinforcement Learning approaches also for continuous state and action spaces, hyperparameter optimization, more precisely learning rate, and discount factor, and a principled way to identify effective reward functions.

References

1. Mala-Jetmarova, H., Sultanova, N., Savic D. (2017). Lost in Optimization of Water Distribution Systems? A literature review of system operations, Environmental Modelling and Software, 93, 209-254.
2. Sterling, M. J. H. and Coulbeck, B., (1975). A dynamic programming solution to the optimization of pumping costs, in Hybrid genetic algorithm in the optimization of energy costs in water supply networks, ICE Proceedings, vol. 59(2), 813–818.
3. Powell, W.B. (2007). Approximate Dynamic Programming: Solving the Curses of Dimensionality. John Wiley and Sons.
4. Ikonen, E., Selek, I., Tervaskanto, M. (2010). Short-term pump schedule optimization using MDP and neutral GA, IFAC Proceedings Volumes, 43(1), 315-320.
5. Ikonen, E., and Bene, J. (2011). Scheduling and disturbance control of a water distribution network. Proc. 18thWorld Congress of the International Federation of Automatic Control (IFAC 2011), Milano, Italy.
6. Sutton, R.S., Barto, A.G., 1998. Reinforcement Learning: An introduction – Adaptive Computation and Machine Learning. MIT Press, Cambridge, USA.
7. Fracasso, P.T., Barnes, F.S., Costa, A.H.R. (2013). Energy cost optimization in water distribution systems using Markov Decision Processes. International Green Computing Conference Proceedings, Arlington, 1-6.
8. Fracasso, P.T., Barnes, F.S., Costa, A.H.R. (2014). Optimized Control for Water Utilities, Procedia Engineering, 70, 678-687.

9. Candelieri, A., Perego, R., Archetti, F. (2018). Intelligent Pump Scheduling Optimization in Water Distribution Networks, In Proceedings of the 12th Learning and Intelligent Optimization Conference (LION), June 10-15 2018, Kalamata, Greece [ahead of printing].
10. Pasha, M. F. K., and Lansey, K. (2009). Optimal pump scheduling by linear programming. In Proceedings of World Environmental and Water Resources Congress 2009 - World Environmental and Water Resources Congress 2009: Great Rivers. Vol. 342, 395-404.
11. Puterman, M. (1994). Markov Decision Processes | Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York, NY.
12. Howard R. A., Dynamic Programming and Markov Processes. Cambridge, USA: MIT Press, 1960.
13. Bellman, R.E., (1957). Dynamic Programming. Princeton University Press, Princeton, USA.
14. Wiering M. and Van Otterlo M., Reinforcement Learning - State-of-the- Art, 1st ed. Berlin, Germany: Springer-Verlag, 2012.
15. Powell, W.B. (2007). Approximate Dynamic Programming: Solving the Curses of Dimensionality. John Wiley and Sons.
16. Ertin, E., Dean, A. N., Moore, M.L. and Priddy., K.L., 2001. Dynamic optimization for optimal control of water distribution systems. Applications and Science of Computational Intelligence IV, Proc. SPIE Vol. 4390, pp. 142-149.